# Enhancing Prediction in Cyclone Separators through Computational Intelligence

Oluwaseyi Ogun
*Process Systems Engineering Laboratory*
*Obafemi Awolowo University*
Ile-Ife, Nigeria
ogun.o.a@keemail.me

Mbetobong Enoh
*Electrical and Electronics Engineering*
*Akwa Ibom State University*
Nigeria
mbetobongenoh@aksu.edu.ng

Georgina Cosma
*School of Science*
*Loughborough University*
United Kingdom
g.cosma@lboro.ac.uk

Aboozar Taherkhani
*School of Science and Informatics*
*De Montfort University*
United Kingdom
aboozar.taherkhani@dmu.ac.uk

Vincenzo Madonna
*PEMC Research Group*
*The University of Nottingham*
Nottingham, UK
Ezzvm2@nottingham.ac.uk

***Abstract*—Pressure drop prediction is critical to the design and performance of cyclone separators as industrial gas cleaning devices. The complex nonlinear relationship between cyclone Pressure Drop Coefficient (PDC) and geometrical dimensions suffice the need for state-of-the-art predictive modelling methods. Existing solutions have applied theoretical/semi-empirical techniques which fail to generalise well, and some intelligent techniques have also been applied such as the neural network which can still be improved for optimal equipment design. To this end, this paper firstly introduces a fuzzy modelling methodology, then presents an alternative Extended Kalman Filter (EKF) for the learning of a Multi-Layer Neural Network (MLNN). The Lagrange dual formulation of Support Vector Machine (SVM) regression model is deployed as well for comparison purposes. For optimal design of these models, manual and grid search techniques are used in a cross-validation setting subsequent to training. Based on the prediction accuracy of PDC, results show that the Fuzzy System (FS) is highly performing with testing mean squared error (MSE) of 3.97e-04 and correlation coefficient (R) of 99.70%. Furthermore, a significant improvement of EKF-trained network (MSE = 1.62e-04, R = 99.82%) over the traditional Back-Propagation Neural Network (BPNN) (MSE = 4.87e-04, R = 99.53%) is observed. SVM gives better prediction with radial basis kernel (MSE = 2.22e-04, R = 99.75) and provides comparable performance to universal approximators. In comparison to conventional theoretical and semi-empirical models, intelligent approaches can provide far better prediction accuracy over a wide range of cyclone designs, while the EKF-MLNN performance is noteworthy.**

***Index Terms*—Cyclone, Pressure drop coefficient, Fuzzy system, Support vector machine, Extended Kalman filter, Multilayer neural network, Cross-validation.**

## I. Introduction

Cyclone separators are air pollution control devices which are widely used in industries that generate gases containing entrained particles, to remove these particles from carrier gas streams before they are discharged. Their design simplicity, low cost and ruggedity which make them suitable for extreme operating conditions, promote their usage and popularity with the industry and more so, the lack of moving parts makes maintenance very cheap. Some industrial applications include the removal of coal dust in power plants, removal of saw dust in sawmills, as spray dryers, etc. However, cyclone separators, like any other technology, require improvement/modification to enhance performance and thus promotes a wider adoption and with the deleterious effect of particulate pollution, it becomes imperative to enhance the performance of pollution control systems [1]. Numerous designs of cyclones exist which are in use for different purposes such as the uniflow, straight-through and reverse flow cyclones. However, the reverse flow is more common which has a tangential rectangular inlet and commonly used for industrial gas cleaning (Fig. 1). Cyclones exploit the centrifugal force generated by the circular spinning of the inlet gas stream at high speed to bring about separation. This then causes the particulate solids to be thrown to the cyclone walls, loose speed and subsequently falls to the bottom where it is discharged. Besides Particle collection efficiency, pressure drop is another major performance metric for cyclone performance evaluation and to this end, accurate mathematical model of the complex relationship between the pressure drop and cyclone dimensions is critical. Several approaches exist in the literature to describe the effect of cyclone geometry on pressure drop. These include: Mathematical models [2] i) theoretical and semi-empirical models, and ii) statistical models; and Computational Fluid Dynamics (CFD) simulations, e.g.,[3][4].

Theoretical/semi-empirical mathematical models based on physical descriptions and detailed understanding of gas flow patterns and energy loss mechanisms in cyclones have been developed by several researchers, including the models of [5][6][7][8][9][10]. However, different simplifying assumptions inherent in these models, and different physical principles being exploited in the modelling can result in significant disparities between predictions and measurements. For example, [11] stated that none of these models predicts pressure drop accurately for a wide range of cyclone designs, and pressure

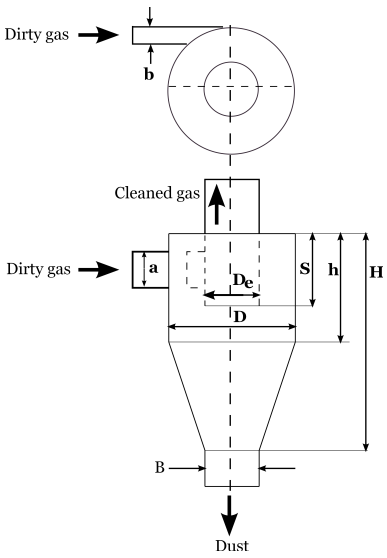


Fig. 1. Schematic of a reverse-flow cyclone separator

drop predictions of some models are twice the measured values [12]. Computational fluid dynamics (CFD) simulations provide a reliable method to examine the effect of design changes on performance and provide an excellent approach to modelling cyclones as they are able to predict fluid flow patterns in great details [13]. However, CFD can be computationally challenging particularly in solving the famous Navier-Stokes equations. While data-driven intelligent methods such as Artificial Neural Network (ANN) [14] and the Least Squares Support Vector Machine (LS-SVM) [2] are promising and have yielded superior modelling capabilities, there is still ample room for improvements, specifically with the choice of learning algorithm, network architecture and hyper-parameters selection/optimisation.

The current state-of-the-art development is hinged on artificial intelligence (AI), simulating natural intelligence to deliver results. The complexity of certain real world phenomena makes traditional methods insufficient to describe them, and the inadequacy of classical computational methods to effectively describe complex nonlinear trends has motivated the need for data-driven intelligent methods which do not require details of the underlying phenomenon, but can effectively extract useful information from domain data for decision making. That said, this paper introduces a fuzzy methodology and a more effective (performance-wise) learning algorithm over the traditional Back-Propagation Neural Network (BPNN), for cyclone design modelling.

The aim of this paper is two-fold: first, to use the dataset from the measurement of ninety-eight cyclone configurations [11] to develop prediction models, and then to draw conclusions from the obtained models for the more suitable approach (based on results) to pressure drop prediction in cyclone separators. The paper is structured as follows: Section II discusses the experimental methodology which include the process of selecting the variables for building the prediction models; Section III develops the mathematical models and algorithms for the prediction models; Section IV presents simulation results with discussion, and comparison of models and finally, conclusions are presented in Section V.

TABLE I
INPUT AND OUTPUT VARIABLES FOR THE CYCLONE

| | | | | **Input** | | | | **Output** |
|---|---|---|---|---|---|---|---|---|
| Variables | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $z$ |
| Spec | De/D | a/D | b/D | S/D | H/D | h/D | B/D | PDC |

## II. Experimental Methods

### A. Model variable selection

Selection of the right response and predictor variables for cyclone modelling is critical to a successful prediction, and the effectiveness/suitability of the consequent model for other purposes will greatly depend on the variables used. Thus, it has been shown that all eight dimensions of a cyclone affect its pressure drop to different extent [11], hence, establishes the predictors. The response (performance) variable in this case is the Pressure Drop Coefficient (PDC).

### B. Pressure drop coefficient (PDC)

The pressure drop, $\Delta P$, is often expressed in terms of PDC, which is a dimensionless quantity and a complex nonlinear function of cyclone geometrical dimensions (i.e., barrel diameter (*D*), total cyclone height (*H*), vortex finder (*De*), vortex finder length (*S*), inlet height (*a*), inlet width (*b*), height of the cylindrical section (*h*), and the cone-tip diameter (*B*), and operating conditions [14]:

$$\Delta P = PDC \left( \frac{\upsilon_i^2 \rho_g}{2} \right) \tag{1}$$

where $\upsilon_i$ (m/s) is the gas inlet velocity, and $\rho_g$ (kg/m$^3$) is the gas density. To facilitate a more accurate determination of *PDC*, all eight dimensions of the cyclone are critical as they affect the pressure drop to different extent [14]. Thus, these dimensions are typically characterised by *D*, and expressed as seven dimensionless geometric ratios as shown below [14]:

$$PDC = f \left( \frac{De}{D}, \frac{a}{D}, \frac{b}{D}, \frac{S}{D}, \frac{H}{D}, \frac{h}{D}, \frac{B}{D} \right) \tag{2}$$

Since *PDC* is a function of cyclone dimension ratios, as shown in (2), it is not affected by operating conditions such as the inlet gas flow rate, and should remain constant for any cyclone configuration, irrespective of size, provided that the dimension ratios remain the same. Thus, $PDC$ can be determined experimentally or theoretically for a particular cyclone, but will be determined more accurately in this work using data-driven techniques that takes into account all the geometrical dimensions of the cyclone, and able to generalise well over a wide range of cyclone designs.

## III. Model Development

### A. Fuzzy system (FS)

A FS, as shown in Fig. 2, is a mapping from $\mathbf{x} \in \mathbb{R}^{\mathrm{n}}$ to $\mathbf{z} \in \mathbb{R}^{\mathrm{m}}$. FSs are knowledge-based systems constructed from human knowledge (an expert in the problem domain) in the form of fuzzy *IF-THEN* rules, and are able to transform a knowledge base into a nonlinear mapping through a systematic procedure as depicted in Fig. 2. The input is a crisp real number in the input universe of discourse, **U**; fuzzifier transforms the crisp input into a membership value in the range of [0, 1], which in turn activates the rules; the inference engine applies the rule base, and i) checks for multiple antecedents and applies appropriate fuzzy operator, e.g., **AND** or **OR**, to obtain a single value for the antecedent, ii) applies implication to modify the output fuzzy set based on the firing strength (output of the antecedent) of each rule, iii) aggregates all modified output fuzzy sets of all the rules into a single fuzzy set; and finally, the aggregated fuzzy set is defuzzified into a single crisp value in the range of the output universe of discourse, **V**.

The heart of a FS is the rule base as it combines all other components to implement the rules [15]. A fuzzy rule base is made up of conditional *IF-THEN* statements:

$$\Re^i : IF\ x_1\ is\ A_1^i\ and\ \cdots\ and\ x_n\ is\ A_n^i,\ THEN\ z\ is\ B^i \quad (3)$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T \in \mathrm{U}$ and $\mathrm{z} \in \mathrm{V}$ comprise the linguistic input and output variables, respectively, $A_j^i$ and $B^i$ are fuzzy sets defined by membership functions over the input and output universes of discourse, $\mathrm{U}_j$ and V, respectively, i = 1, 2, $\cdots$, $\Re$, and $\Re$ is the number of rules in the rule base. The *IF* part of the rule i.e., $x_1\ is\ A_1^i\ and\ \cdots\ and\ x_n\ is\ A_n^i$, is called the antecedent, while the *THEN* part i.e., $y\ is\ B^i$, is called the consequent. This is a case of multi-input single-output system. In depth studies on fuzzy logic and FS can be found elsewhere, e.g., [15][16][17][18].

A FS is thus trained to learn a complex input-output mapping from the geometrical measurements of ninety-eight cyclones dataset in [11]. Depending on the type of inference engine, fuzzifier, and defuzzifier used, different combinations of these three modules can result in different FS, however, not all of these combinations make much sense [15]. Here, the universal approximation capability of a FS is exploited, which is smooth and continuous, and able to interpolate well between data points and therefore suitable for the current purpose. Hence, a FS with rule base as in (3), product inference engine, singleton fuzzifier, centre average defuzzifier, and Gaussian membership functions, as shown in (4), is a design choice.

$$g(\mathbf{x}|\theta) = \frac{\sum_{i=1}^{\Re} b_i \prod_{j=1}^{n} \exp(-\frac{1}{2}(\frac{x_j - c_j^i}{\sigma_j^i})^2)}{\sum_{i=1}^{\Re} \prod_{j=1}^{n} \exp(-\frac{1}{2}(\frac{x_j - c_j^i}{\sigma_j^i})^2)} \quad (4)$$

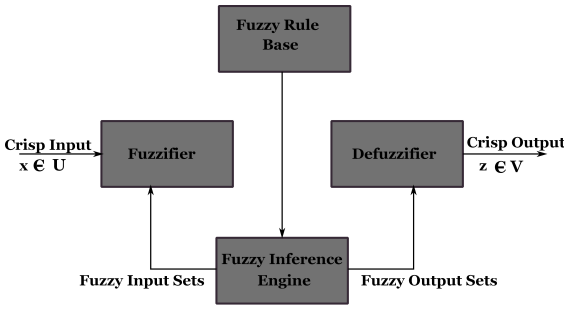


Fig. 2. Schematic diagram of a fuzzy system

$$\theta = [b_1,\ \cdots,\ b_{\Re},\ c_1^1,\ \cdots,\ c_n^1,\ \cdots,\ c_1^{\Re},\ \cdots,\ c_n^{\Re},\ \sigma_1^1,\ \cdots,\ \sigma_n^1,\ \cdots,\ \sigma_1^{\Re},\ \cdots,\ \sigma_n^{\Re}]^T$$

where $b_i$ is the centre of the output fuzzy set for the $ith$ rule, $c_j^i$ is the centre point of the $jth$ input membership function for the $ith$ rule, $\sigma_j^i > 0$ is the width (or spread) of the membership function for the $jth$ input and the $ith$ rule, and $\theta$ is the design vector. The FS (4) is designed when the system parameters in $\theta$ are determined.

### B. Gradient descent learning

In order to successfully design (4) and construct a FS that can interpolate well between data points, the gradient descent algorithm is presented to tune all the parameters in $\theta$. The aim of gradient methods is to minimise the squared error, $e_l$, in (5) for each training data pair, $l$, by choosing $\theta$.

$$e_l = \frac{1}{2}[g(\mathbf{x}^l|\theta) - z^l]^2 \quad (5)$$

Thus, the gradient descent update law for the Gaussian input centres, $c_j^i$ $(i = 1, 2, \cdots, \Re,\ j = 1, 2, \cdots, n)$, is given as

$$c_{j,t+1}^i = c_{j,t}^i - \lambda \frac{\partial e_l}{\partial c_j^i}\bigg|_t + \beta(c_{j,t}^i - c_{j,t-1}^i) \quad (6)$$

where the subscripts $t+1$, $t$ and $t-1$ denote the future and past values, $\lambda > 0$ is the learning rate, and $\beta > 0$ is the momentum factor. Chain rule of calculus is applied to $\frac{\partial e_l}{\partial c_j^i}$ at time $t$, making references to (4) and (5).

$$\frac{\partial e_l}{\partial c_j^i} = e_{l,t} \frac{\partial g(\mathbf{x}^l|\theta_t)}{\partial \mu_i(\mathbf{x}^l, t)} \frac{\partial \mu_i(\mathbf{x}^l, t)}{\partial c_j^i} \quad (7)$$

so that

$$\frac{\partial g(\mathbf{x}^l|\theta_t)}{\partial \mu_i(x^l, t)} = \frac{b_{i,t} - g(\mathbf{x}^l|\theta_k)}{\sum_{i=1}^{\Re} \mu_i(\mathbf{x}^l, t)} \quad (8)$$

and

$$\frac{\partial \mu_i(\mathbf{x}^l, t)}{\partial c_j^i} = \mu_i(\mathbf{x}^l, t) \left( \frac{\mathbf{x}_j^l - c_{j,t}^i}{(\sigma_{j,t}^i)^2} \right) \quad (9)$$

Substituting (8) and (9) into (7) and plugging the result into (6) gives the update equation for the input Gaussian centres, where

TABLE II
STATISTICAL SUMMARY OF THE 98 CYCLONES DATASET IN [11]

| | De/D | a/D | b/D | S/D | H/D | h/D | B/D | PDC |
|---|---|---|---|---|---|---|---|---|
| Min | 0.25 | 0.113 | 0.067 | 0.39 | 1.158 | 0.501 | 0.14 | 2.3 |
| Mean | 0.428 | 0.630 | 0.211 | 0.891 | 3.283 | 1.189 | 0.342 | 23.268 |
| Standard dev. | 0.1104 | 0.2618 | 0.0936 | 0.4289 | 2.0956 | 0.6729 | 0.1498 | 32.8858 |
| Max | 0.667 | 1.0 | 0.4 | 3.052 | 10.97 | 3.5 | 1.0 | 155.3 |

$$\mu_i(\mathbf{x}^l, t) = \prod_{j=1}^{n} \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}_j^l - c_{j,t}^i}{\sigma_{j,t}^i}\right)^2\right) \quad (10)$$

Similarly, the gradient update laws for $b_i$ and $\sigma_j^i$ can be obtained. Here, only the number of rules, $\Re$, in the rule base needs to be set before learning begins, while the system parameters in $\theta$ are initialised to some values and tuned to optimum in the course of training. In gradient descent learning, it is necessary to ensure that learning converges before training is finished, thus, longer epochs might be required to achieve this, or the network error is compared to some pre-specified error goal during training and if sufficiently small, training is finished.

### C. Multi-layer neural network (MLNN)

The MLNN shown in Fig. 3 is a feed-forward network architecture composed of three layers: input, hidden and output layers. The inputs, $x_1, \cdots, x_7$, are each weighted and passed on to each of the $S$ neurons in the hidden layer. Each neuron has a bias of constant unit input which is summed with the weighted inputs to form the net input that goes into each neuron's transfer function. The $S$ outputs from the hidden layer are again weighted, summed with a bias and *squashed* by the transfer function of the output layer neuron to give the overall output of the network. According to the universal approximation theory, a single hidden layer network of this kind is able to approximate any non-linearity to arbitrary accuracy, given sufficient number of neurons in the hidden layer. Thus, the output, $O_i^{m+1}$, for any hidden neuron of layer $m+1$ is given by

$$O_i^{m+1} = f_i^{m+1}\left\{\sum_{j=1}^{S^m} w_{i,j}^{m+1} a_j^m + b_i^{m+1}\right\}; \quad m = 0, 1, 2, \cdots, M-1 \quad (11)$$

where $M$ is the last layer of the network, $f_i^{m+1}$ is the transfer function of the $ith$ neuron of the $(m+1)th$ layer, $a_j^m$ is the output of $jth$ neuron of layer $m$, and $b_i^{m+1}$ is the bias term associated with the $ith$ neuron of the $(m+1)th$ layer. The designation $w_{i,j}$ is interpreted as the weight connection from input *j* to neuron *i*. In a multi-layer feedforward network, the output of the previous layer becomes the input to the next layer, hence, for the considered two-layer network (excluding the input layer which is just a pass in layer) of Fig. 3, the input vector, $\mathbf{x}$, is vectorised, $\mathbf{x} = [x_1, \cdots, x_7]^T$, and the overall network equation (12) can be constructed, where the input to the last layer is obtained from (11) and the last layer output equation is obtained by setting $m$ = $M-1$ in (11).

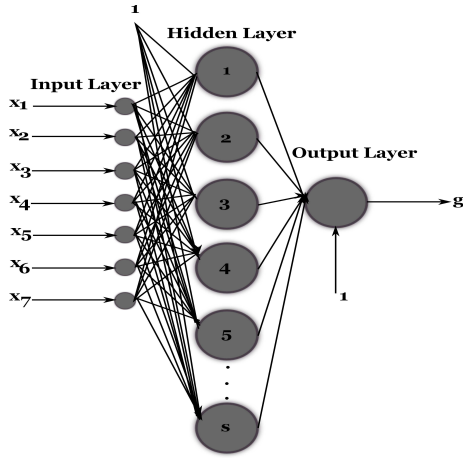


Fig. 3. A three-layer neural network architecture with seven inputs, S hidden neurons and one output neuron

$$g = f_1^2\left\{\sum_{j=1}^{S^1} w_{1,j}^2 . f_j^1\left\{\sum_{r=1}^{R} w_{j,r}^1 \mathbf{x}(r) + b_j^1\right\} + b_1^2\right\}. \quad (12)$$

Where $R$ is the dimension of $\mathbf{x}$ which is seven in this case, and $\mathbf{x}(r)$ is the $rth$ element of $\mathbf{x}$. As the input is propagated forward through the network, the output is compared with the target value and a back-propagation learning rule is then used to adjust the weights and biases of the network in order to move the output closer to the target. This is the learning phase. The experiential knowledge gained during training is captured in the network weights and biases which hold the domain solution. Shortly following the development of back-propagation algorithms for neural networks in the 1980s came the use of extended Kalman filter (EKF) [19][20] and afterwards, the unscented Kalman filter (UKF) [21], for neural training. The EKF was shown to outperform the traditional back-propagation algorithm on classification problems [19], and is considered in this paper for modelling purposes.

### D. Neural weight learning

The learning problem involves the determination of the nonlinear mapping

$$g_t = \Gamma(\mathbf{x}_t, \mathbf{w}) \quad (13)$$

where the subscript *t* denotes the current time instance, and weight vector, $\mathbf{w}$, is composed of the parameters to be learned. The mapping error, $e_t$, is the difference between the

target output, $z_t$, and the network output, $g_t$, i.e., $e_t = z_t - g_t$, and the goal of learning is to find $\mathbf{w}$ which minimises some specified performance criterion. To achieve this, a state-space representation of (13) is first written as

$$\begin{aligned} \mathbf{w_{t+1}} &= \mathbf{w_t} + \mathbf{r_t} \\ d_t &= \Gamma(\mathbf{x}_t, \mathbf{w_t}) + e_t, \end{aligned} \tag{14}$$

where $\mathbf{w_t}$ is a stationary process with identity state transition matrix driven by process noise $\mathbf{r_t}$, and $e_t$ is the measurement noise. Since $\Gamma(.)$ is a nonlinear neural network in this case, it cannot be used directly in the EKF algorithm. Instead, the partial derivatives with respect to network weights are computed and at each time instance $t$, these derivatives are evaluated at the current predicted state (weight estimates) which are then used in the EKF equations.

*1) Recursive EKF algorithm:* For EKF implementation, the neural network weight connections including the connections from input to hidden, and hidden to output layers, are coalesced in a single state vector as a linear array. Thus, for the network of Fig. 3, the corresponding Kalman state vector would be written as $\mathbf{w} = [w^1_{1,1}, \cdots, w_{S,R},\ \ b^1_1, \cdots, b^1_S,\ \ w^2_{1,1}, \cdots, w^2_{1,S},\ \ b^2_1]^T$. Algorithm 1 presents the EKF recursive equations which comprise two steps: prediction and correction. The prediction step forms the predictor for the nex t observation, and the result is the a priori estimates at time *t*, while the correction step updates the a priori estimates from the prediction with new information arriving at time *t*. This result is the a posteriori estimates.

**Algorithm 1** EKF equations

Initialisation:
$\hat{\mathbf{w}}_0 = \mathbb{E}[\mathbf{w}]$
$\mathbf{P}_0 = \mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}}_\mathbf{0})(\mathbf{w} - \hat{\mathbf{w}}_\mathbf{0})^T]$

**for** t = 1 : $N_s$ **do** ▷ Loop over training samples

▷ Prediction/time update

$\hat{\mathbf{w}}_{t|t-1} = \hat{\mathbf{w}}_{t-1}$
$\mathbf{P}_{t|t-1} = \mathbf{P}_{t-1} + \mathbf{Q}$

▷ Correction/measurement update

$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t(R + \mathbf{H}_t^T\mathbf{P}_{t|t-1}\mathbf{H}_t)^{-1}$
$\hat{\mathbf{w}}_{t|t} = \hat{\mathbf{w}}_{t|t-1} + \mathbf{K}_t(z_t - \Gamma(\mathbf{x}_t, \hat{\mathbf{w}}_{t|t-1}))$
$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t^T)\mathbf{P}_{t|t-1}$

In Algorithm 1, $\mathbf{Q}$ is the process noise covariance, $R$ is the measurement noise variance, $\mathbf{H}_t$ is the first order linearisation of $\Gamma(.)$, evaluated at the current weight estimates,i.e., $\mathbf{H}_t = \left.\frac{\partial\Gamma(.)}{\partial\mathbf{w}}\right|_{\mathbf{w}_t}$, $\mathbf{P}$ is the error covariance of the network weights, and $t|t-1$ denotes the estimate at time $t$ using the available information up to and including the time instance $t-1$.

To begin the recursion, $\mathbf{w}(0)$ is defined by a Gaussian distribution with $\mathcal{N}(\bar{\mathbf{w}}(0), \mathbf{P}(0))$, where $\bar{\mathbf{w}}(0)$ and $\mathbf{P}(0)$ capture any a priori knowledge about the network weights. However, where such knowledge is not available, $\bar{\mathbf{w}}(0)$ can be initialised from a random distribution and $\mathbf{P}(0) = \mu\mathbf{I}$, where $\mu$ is a large number, e.g., $10^4$.

### *E. Support Vector Machine (SVM)*

SVM, first identified by [22], was originally developed to solve classification problems, but has found successful applications in function approximation and time series prediction problems [23][24]. To configure SVM for regression, the linear $\epsilon$-insensitive loss function (15) is introduced, and the goal here is to find a function $g(x)$ that deviates from the true response by no greater than $\epsilon$ for each training instance, and be as flat as possible.

$$L_\epsilon = \begin{cases} 0 & \text{if } |z - g(\mathbf{x})| \le \epsilon \\ |z - g(\mathbf{x})| - \epsilon & \text{otherwise} \end{cases} \tag{15}$$

Equation (15) simply means that if the regression error lies within the $\epsilon$-tube, the loss is taken to be zero. If otherwise, the loss is equal to the difference between the regression error and $\epsilon$, where $\epsilon$ specifies the desired approximation accuracy. SVM regression problems are easily solved in their Lagrange dual formulation, and to obtain the dual formula for $\epsilon$-insensitive support vector regression, non-negative Lagrange multipliers, $\alpha$ and $\alpha^*$, are assigned to each training instance, $\mathbf{x}$, and the Karush-Kuhn-Tucker (KKT) first order conditions for optimality are applied. A dual problem results which finds $\alpha$ and $\alpha^*$ that maximises the following quadratic objective function:

$$\begin{aligned} \max_{\alpha,\alpha^*} \quad & -\frac{1}{2}\sum_{i=1}^{N_s}\sum_{j=1}^{N_s}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathbf{x}_i'\mathbf{x}_j - \\ & \epsilon\sum_{i=1}^{N_s}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N_s} z_i(\alpha_i - \alpha_i^*) \\ \text{subject to :} \quad & \\ & \sum_{t=1}^{N_s}(\alpha_i - \alpha_i^*) = 0 \\ & 0 \le \alpha_i \le C \\ & 0 \le \alpha_i^* \le C \end{aligned} \tag{16}$$

The KKT conditions allow each training point to be classified into support vector types. For non-linear regression problems as is the case in this paper, (16) is kernelised by replacing the linear dot product $\mathbf{x}_i'\mathbf{x}_j$ with a non-linear kernel function, $G(\mathbf{x}_i, \mathbf{x}_j)$, such as the radial basis function. Thus, the approximate prediction function for the SVM regression problem is written as:

$$g(\mathbf{x}) = \sum_{t=1}^{Ns}(\alpha_t - \alpha_t^*)G(\mathbf{x}_t, \mathbf{x}) + b \tag{17}$$

## IV. Experimental Methodology, Results and Discussion

### *A. Cyclone dataset*

The cyclone dataset [11] is composed of the measurements of pressure drop for ninety-eight different cyclones gathered

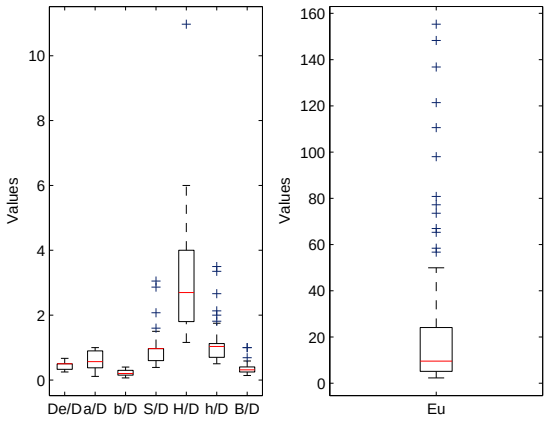


Fig. 4. Statistical distribution of the cyclone dataset

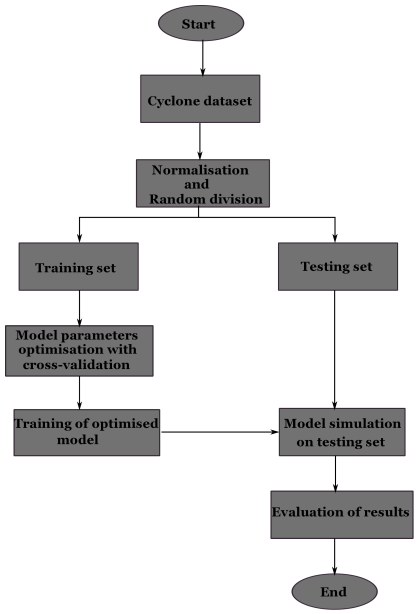


Fig. 5. Procedure used for models development

from different literature sources. Each cyclone measurement data is composed of seven geometrical dimensionless ratios, and the corresponding PDC. The decision to include a cyclone data in the dataset was based on certain four criteria being met (see [11]). Due to the unavailabilty of accurate physics-based nonlinear model of cyclone separator dynamics, it is difficult to synthesise more data for model training. Thus, cross-validation is employed for effective training and generalisation of the intelligent models using the limited available data.

Each column of Table II shows the statistics of each variable. It is observed from the table that there is a large difference in the order of magnitudes between the variables, and *Eu* is noticeably widely dispersed and its mean cannot be said to be representative of the central value considering the range of the data. A more informative and graphical approach to viewing the distribution of the data is with a box and whisker plot. Fig. 4 shows the box plot for each variable, representing information from a five-number summary: minimum value, first quartile, median, third quartile and the maximum value. The red horizontal lines in the boxes denote the median values, the ends of the boxes represent the lower and upper quartiles, and the whiskers are the two lines outside the boxes, extending to the maximum and minimum observations. The data are mostly skewed as the medians cut the boxes into two unequal parts, and the statistics of the variables are significantly different from one another. *S/D*, *h/D* and *Eu* have significant number of outliers as indicated by the '+' symbols. *De/D*, *S/D* and *h/D* have data that are more condensed (closer together) around the larger values (*De/D* and *S/D* have a couple of points with same values as the median/upper quartile since these two coincide), *b/D* and *Eu* are more condensed around the smaller values of the variables. *H/D*, *a/D* and *B/D* are roughly symmetrical (evenly spread out) about their medians.

### B. Data pre-processing

The dataset obtained from [11] and described in the previous section, is the only comprehensive experimental dataset on cyclone that was put together from different literature sources, and is used in the present study for pressure drop prediction through intelligent modelling. To facilitate effective training of the intelligent models, each variable in the dataset is first normalised in the range of [0, 1] according to (18) due to the large difference in the order of magnitude, and subsequently randomly divided into 80% ($\approx$ 78 samples) training and 20% ($\approx$ 20 samples) testing sets.

$$z_n = \frac{z_i - z_{min}}{z_{max} - z_{min}} \tag{18}$$

Where $z_n$ is the normalised value of the observed variable, $z_i$ is the actual variable, $z_{min}$ and $z_{max}$ are the minimum and maximum values, respectively, of the observed variable. Both the input and output variables are subjected to the normalisation procedure.

To convert simulation results back to the original scale, de-normalisation is performed using the same normalisation procedure in (18). The processing function essentially becomes an integral part of the models.

### C. Hyper-parameters selection/optimisation

In the fuzzy model, the only parameter available for tuning is the the number of rules. Thus, this is manually tuned with $V$-fold cross-validation, where $V$ = 10 in this case, to prevent over-fitting. In a $V$-fold cross-validation experiment, the original sample is randomly divided into $V$ folds, where a single fold is used for independent testing and the other $V-1$ folds are used to train the model. This process is repeated $V$ times, with each of the $V$ folds being used exactly once for testing. The cross-validation error is the average mean squared error (MSE) on the $V$ testing folds. Thus, $\Re$ = 29 rules is found optimal. Similarly, the number of hidden neurons, $N_h$, influences the performance of neural networks. Several

rules of thumb exist in the literature to fix the size of the hidden layer with each resulting in different values on the same application (an indeterminate situation), and none of which guarantees optimality in any one application. Thus, the search for hidden neurons is an optimisation problem. Applying the same procedure, $N_h$ = 14 with hyperbolic tangent neurons.

The search for optimal SVM parameters employs a grid search technique with 10-fold cross-validation for the regularisation parameter, $C$, the radial basis kernel scale factor, $K_s$, and $\epsilon$. Optimal search for these parameters is performed in the range of $C$ = [0.001, 1000], $K_s$ = [0.001, 1000] and $\epsilon$ = [1e-04, 8.24]. And the optimal values found are 1000, 2.154 and 3.82e-03, respectively.

### *D. Model training*

The optimal models, based on the parameters obtained previously, are trained on the entire dataset used for cross-validation, i.e., all the $V$-folds are put together to design the final prediction models and the abilities of the trained models to generalise on unseen future data are determined on the remaining 20% testing set, held out for independent testing purposes. Fig. 5 sets out the training/testing procedure used in this work. Prior to training, the choice of initial parameters is critical to the success of gradient descent algorithms. The closer the initial values are to the optimum, the higher the chance of converging on the optimum. Thus, FS parameters are initialised with the first $\Re$ training instances, i.e., $c_j^i(0) = x_{j,0}^i$, $b_i(0) = z_{i,0}$, and $\sigma_j^i(0)$ = [max($x_{j,0}^i : i = 1, 2, \cdots, \Re$) – min($x_{j,0}^i : i = 1, 2, ..., \Re$)]/$\Re$, where $i = 1, 2, \cdots \Re$, $j = 1, 2, \cdots, n$. And the neural network weights, $\mathbf{w}(\mathbf{0})$, are drawn from a random distribution in the [0, 1] range and $P(0) = 1e4\mathbf{I}$. Ideally, $\mathbf{w}(\mathbf{0}) = 0$ is appropriate, random initialisation did produce much better response in this application.

To ensure that the learned weights converge, Figs 7 and 8 show the weight learning curves for the EKF and gradient descent training, respectively. EKF stabilises at the 143rd epoch and after which, learning only proceeds very slowly, while gradient descent requires much longer epochs to converge to the result obtained. The back-propagation used here is based on the momentum gradient descent algorithm, and comparing EKF to back-propagation, the following are observed: i) BPNN takes longer epochs (about 10 times longer than that of EKF-MLNN) to converge to the results presented in Table III, ii) 21 hidden neurons are optimal for BPNN to achieve the mapping accuracy as compared to 14 needed by EKF-MLNN, iii) EKF, however, requires more parameters to be tuned/initialised to get excellent result. Fig. 6 shows competitive results from all models. However, if higher prediction accuracy is desired, the EKF-trained MLNN may be a suitable choice but trade-offs might be necessary in practice. Furthermore, this study has considered the radial basis kernel for the SVM model after comparison (following the same procedure as in Fig. 5) with polynomial (MSE = 1.3e-03, R = 98.52%) and linear (MSE = 1.4e-02, R = 82.1%) kernels.

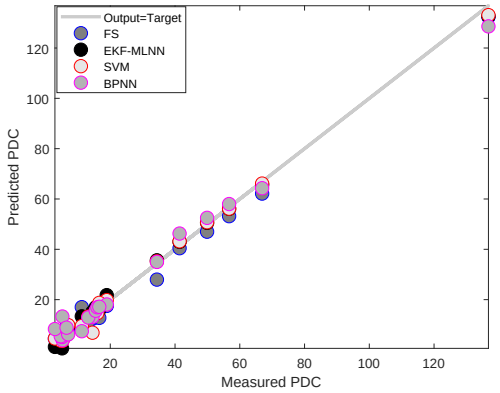


Fig. 6. Comparison of the AI models for PDC prediction

TABLE III
MODELS TESTING PERFORMANCE

| | **FS** | **EKF-MLNN** | **SVM** | **BPNN** |
|---|---|---|---|---|
| Correlation coefficient (R) | 0.9970 | 0.9982 | 0.9975 | 0.9958 |
| Mean squared error (MSE) | 3.97e-04 | 1.62e-04 | 2.22e-04 | 4.89e-04 |

### *E. Comparison with theoretical, semi-empirical and multi-regression models*

The EKF-MLNN model is compared to conventional cyclone pressure drop models developed by [9][25][26]. It is very clear from Fig. 9 that EKF-MLNN clearly outperforms the conventional theoretical, semi-empirical and multi-regression models, and the smallest MSE in Table V is about 45 times larger than that of MLNN. The model of Shephered & Lapple [9] is surprisingly better than others in terms of MSE and R values because this model does not take into account the effect of all cyclone dimensions. However, this may be attributed to the nature of the data used for testing. It is noteworthy that none of the conventional models considered matches the accuracy of any of the intelligent methods.

## V. CONCLUSIONS

In this paper, three AI models: fuzzy system, multi-layer neural network and support vector machine configured for regression, have been developed and compared to obtain

TABLE IV
MODELS PARAMETERS AND SIMULATION VALUES

| **FS** | **MLNN** | **SVM** |
|---|---|---|
| $\lambda$ = 0.01 | $R$ = 1.1 | $C$ = 1000 |
| $\beta$ = 0.72 | $Q$ =1e-03$\mathbf{I}$ | $K_s$ = 2.154 |
| $\Re$ = 29 | $N_h$ = 14 | $\epsilon$ = 3.82e-03 |

TABLE V
PERFORMANCE OF CONVENTIONAL MODELS TO EKF-MLNN

| | **MSE** | **R** |
|---|---|---|
| Casal and Martinez-Benet[26] | 1.23e-01 | 0.9441 |
| Shepherd and Lapple[9] | 7.3e-03 | 0.9788 |
| Stairmand[25] | 1.4e-02 | 0.9399 |
| EKF-MLNN | 1.62e-04 | 0.9982 |

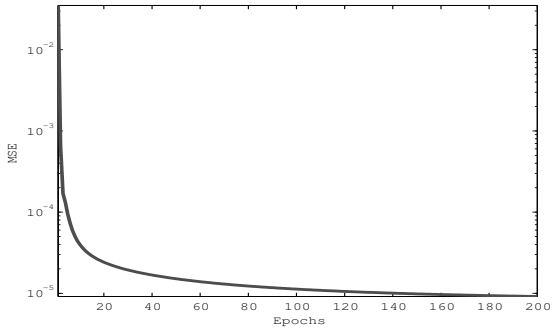


Fig. 7. Convergence of the EKF weights learning

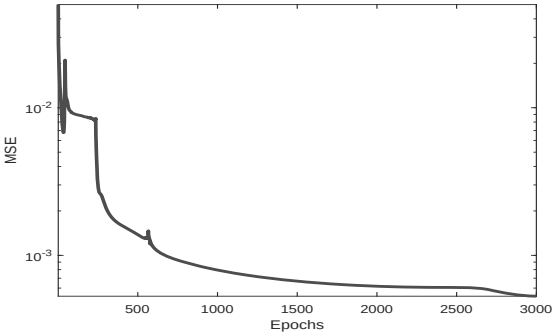


Fig. 8. Convergence of the gradient descent weights learning

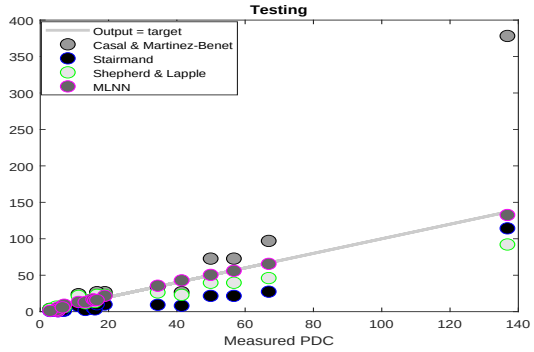


Fig. 9. Comparison of MLNN with conventional cyclone pressure drop models

a more accurate approach to pressure drop prediction in cyclone separators. Models hyper-parameters are first determined manually and by grid search (for SVM) in a $V$-fold cross-validation setting to prevent over-fitting. The obtained optimal models are subsequently trained and validated on previously unseen data. Although the EKF-trained MLNN slightly outperforms the fuzzy model and SVM, this study has demonstrated that any of these models is suitable for prediction in practice. Furthermore the EKF and the back-propagation algorithm are compared and EKF is shown to be better in terms of early convergence, smaller hidden layer size and prediction accuracy. In comparison with conventional pressure drop models, the EKF-MLNN model is able to achieve a maximum reduction error of about 99% on testing samples, and thus has proven to be able to predict pressure drop more accurately over a wide range of cyclone designs and the same report applies to other AI models. In the future, it is envisaged that the best performing PDC model obtained in this paper and an efficiency model, will be used to design an optimal cyclone geometry/configuration that maximises cyclone performance.

## REFERENCES


[1] L. Wang, M. D. Buser, C. B. Parnell, and B. W. Shaw, "Effect of air density on cyclone performance and system design," *Transactions of the ASAE*, vol. 46, no. 4, p. 1193, 2003.

[2] B. Zhao, "Modeling pressure drop coefficient for cyclone separators: a support vector machine approach," *Chemical Engineering Science*, vol. 64, no. 19, pp. 4131–4136, 2009.

[3] W. Griffiths and F. Boysan, "Computational fluid dynamics (cfd) and empirical modelling of the performance of a number of cyclone samplers," *Journal of Aerosol Science*, vol. 27, no. 2, pp. 281–304, 1996.

[4] J. Gimbun, T. Chuah, A. Fakhru'l-Razi, and T. S. Choong, "The influence of temperature and inlet velocity on cyclone pressure drop: a cfd study," *Chemical Engineering and Processing: Process Intensification*, vol. 44, no. 1, pp. 7–12, 2005.

[5] I. Karagoz and A. Avci, "Modelling of the pressure drop in tangential inlet cyclone separators," *Aerosol Science and Technology*, vol. 39, no. 9, pp. 857–865, 2005.

[6] W. Barth, "Design and layout of the cyclone separator on the basis of new investigations," *Brenn. Warme Kraft*, vol. 8, no. 1, p. 9, 1956.

[7] A. Avci and I. Karagoz, "Theoretical investigation of pressure losses in cyclone separators," *International communications in heat and mass transfer*, vol. 28, no. 1, pp. 107–117, 2001.

[8] J. Chen and M. Shi, "A universal model to calculate cyclone pressure drop," *Powder technology*, vol. 171, no. 3, pp. 184–191, 2007.

[9] C. Shepherd and C. Lapple, "Flow pattern and pressure drop in cyclone dust collectors cyclone without intel vane," *Industrial & Engineering Chemistry*, vol. 32, no. 9, pp. 1246–1248, 1940.

[10] C. J. Stairmand, "The design and performance of cyclone separators," *Trans. Instn. Chem. Engrs.*, vol. 29, pp. 356–383, 1951.

[11] G. Ramachandran, D. Leith, J. Dirgo, and H. Feldman, "Cyclone optimization based on a new empirical model for pressure drop," *Aerosol Science and Technology*, vol. 15, no. 2, pp. 135–148, 1991.

[12] P. K. Swamee, N. Aggarwal, and K. Bhobhiya, "Optimum design of cyclone separator," *AIChE journal*, vol. 55, no. 9, pp. 2279–2283, 2009.

[13] W. Griffiths and F. Boysan, "Computational fluid dynamics (cfd) and empirical modelling of the performance of a number of cyclone samplers," *Journal of Aerosol Science*, vol. 27, no. 2, pp. 281–304, 1996.

[14] B. Zhao and Y. Su, "Artificial neural network-based modeling of pressure drop coefficient for cyclone separators," *chemical engineering research and design*, vol. 88, no. 5-6, pp. 606–613, 2010.

[15] L.-X. Wang, *A course in fuzzy systems*. Prentice-Hall press, USA, 1999.

[16] R. Kruse and C. Moewes, "Fuzzy systems," *BG Teubner Stuttgart*, 1993.

[17] L. A. Zadeh, *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, vol. 6. World Scientific, 1996.

[18] D. J. Dubois, *Fuzzy sets and systems: theory and applications*, vol. 144. Academic press, 1980.

[19] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended kalman algorithm," in *Advances in neural information processing systems*, pp. 133–140, 1989.

[20] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended kalman filter training of feedforward layered networks," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 1, pp. 771–777, IEEE, 1991.

[21] E. A. Wan, R. Van Der Merwe, and A. T. Nelson, "Dual estimation and the unscented transformation," in *Advances in neural information processing systems*, pp. 666–672, 2000.

[22] V. N. Vapnik, *The nature of statistical learning theory*. Heidelberg, DE: Springer science Verlag, 1995.

[23] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transaction on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

[24] L. Cao, H. Lee, C. Seng, and Q. Gu, "Saliency analysis of support vector machines for gene selection in tissue classification," *Neural Computing and Applications*, vol. 11, no. 3–4, pp. 244–249, 2003.

[25] C. Stairmand, "Pressure drop in cyclone separators," *Industrial and Engineering Chemistry*, vol. 16, no. B, pp. 409–411, 1949.

[26] J. Casal, "A better way to calculate cyclone pressure drop," *Chem. Eng.*, pp. 90–99, 1983.